Int. J. Communications, Network and System Sciences, **2012, 5, 839-849**
http://dx.doi.org/10.4236/ijcns.2012.512089 Published Online December 2012 (http://www.SciRP.org/journal/ijcns)

Towards Cloud to Device Push Messaging on Android: Technologies, Possibilities and Challenges

Jarle Hansen[1], Tor-Morten Grønli[1,2], Gheorghita Ghinea[1,2]
[1]School of Information Systems, Computing and Mathematics, Brunel University, London, UK
[2]The Norwegian School of Information Technology, Oslo, Norway
Email: jarle@jarlehansen.net, tmg@nith.no, george.ghinea@brunel.ac.uk

Received October 5, 2012; revised November 2, 2012; accepted November 14, 2012

ABSTRACT

In this paper we look at different push messaging alternatives available for Android. Push messaging provides an important aspect of server to device communication, and we specifically focus on the integration of cloud computing with mobile devices through the use of push-based technologies. By conducting a benchmarking test, we investigate the performance of four relevant push technologies for the Android platform, namely C2DM, XMPP, Xtify and Urban Airship. The comparison focuses on three aspects of the libraries: 1) The stability; 2) Response times; and 3) Energy consumption. The test is conducted on both WLAN and 3G, and includes several mobile device types. Additionally, we also integrate with the Google App Engine to provide the cloud integration server that is responsible for sending push messages to the mobile devices.

Keywords: Cloud Computing; Android; Push Messaging; Cloud Integration; C2DM; XMPP; Urban Airship; Xtify

1. Introduction

The goal of cloud computing is to provide the appearance of unlimited scalability and storage for less money than the in-house data centers [1]. Its success is based on an economy of scale and the relative ease of administration of services, as an entire cloud-based data center could be configured through, for example, a series of web pages. Many businesses are today looking at cloud computing as a viable and cost-effective alternative to hosting their own data centers internally, with large IT companies like Microsoft, Google, and IBM all having initiatives relating to cloud computing [2]. Popular cloud-based platforms include *Microsoft Azure*, *Amazon EC2* and the *Google App Engine*.

In 2011 Gartner reported that cloud computing, and mobile applications and media tablets are on the top 10-list of strategic technologies [3]. In this paper we will investigate how these technologies can cooperate through push messaging, where a content provider publishes information to a subscriber. We have focused on combining cloud computing with mobile applications through the use of push technology on the Android platform.

Before the platform support for push messaging was added to Android it was common to use a polling mechanism. This worked by making the application constantly poll the server for updates. There are several drawbacks with this alternative, especially the challenge of configuring the frequency of poll-requests sent. Another possibility is to push messages using SMS (Short Message Service). Android is able to receive and intercept SMS messages, but they come with their own limitations like availability, cost and message size.

In this paper we present our experience from working closely with push messaging technologies. Specifically, we compare different technologies available for the Android platform, from the standard library provided by Google to commercial options. In total we will look at four alternatives that all provide similar push messaging features. We believe this gives an in-depth look at the state of the art in integration between cloud computing and the Android platform not found in existing research. On the server side we have used the Google App Engine as the cloud-based platform.

We focused on push messaging on Android because it is an important aspect for application developers. Push messaging is in many situations a vital aspect of the usability and functionality of an application. Additionally, as stated by Gartner [4], the popularity of the Android platform means that this area of research highlights challenges that affect a considerable amount of software developers.

Accordingly, the main contribution of our research is to investigate the following topic: *Compare four push-messaging technologies for Android, which are integrated with a cloud-computing environment, in regards to the stability of responses, response times and energy consumption.*

The paper is organised as follows: We begin with a look at related work before presenting a short introduction of the investigated technologies. A description and review of the benchmarking test is shown, and finally the conclusion is presented towards the end.

2. Related Work

Cloud computing is becoming increasingly popular. Features like elasticity, scalability and a new cost-model are providing new and interesting opportunities for many companies. It has proven particularly useful for small and medium enterprises that have a large variation in their computing needs [5]. However, not all businesses will benefit from moving their data centres to the cloud, e.g. when there are government regulations not allowing sensitive data to be stored with an external cloud provider [6].

In an attempt to help decision makers identify their concerns with moving all or parts of their computing needs to the cloud, Khajeh-Hosseini *et al*. [7] has created a *Cloud Adoption Toolkit*. With this toolkit, they argue that one can identify the potential benefit or drawback from moving the IT infrastructure and applications to a cloud provider.

When selecting a cloud platform, there are three main service models to select from [8]:

1) **Software as a Service (SaaS)**, the consumer uses the cloud provider's applications running on a cloud infrastructure;

2) **Platform as a Service (PaaS)**, the consumer is able to deploy (either customer created or acquired) applications onto the cloud infrastructure. The consumer does not manage or control the cloud platform/infrastructure;

3) **Infrastructure as a Service (IaaS)**, the consumer can provision processing, storage, networks, and other computing resources that can be utilised to deploy and run the applications.

In our work we focus on the Google App Engine, which is a PaaS service model making it possible for developers to run their own applications on Google's infrastructure [9]. We used the Google App Engine because it provides good support for the technologies we wanted to test, such as a close integration with C2DM and XMPP, and also a simple and easy administration feature.

The Google App Engine is a cloud-based PaaS service, the platform is pre-configured by Google and provides a much higher abstraction than an IaaS (Infrastructure as a Service) platform like Amazon EC2, where almost the entire stack from kernel and upwards can be controlled [10]. The Google App Engine is also well integrated with other Google services like e-mail and authentication. The platform imposes certain limitations on the developers, for example that threads cannot outlive the request that creates it and a limit of 10 concurrent request threads [11]. However, by enforcing these Google is able to provide very high scalability [9].

In work more closely related to ours, Minstrel [12] has been developed to provide a push-based messaging system. The system utilises the publish/subscribe paradigm and has been extended to support mobile devices. Minstrel and the standard push messaging library for Android, which is called Cloud to Device Messaging (C2DM), have several similarities in how they are built, including the publish/subscribe model, where subscribers must register at the content publisher to receive the messages.

In similar research, the Bakabs application, created for Android and iOS, Paniagua *et al*. [13] use C2DM and the Apple Push Notification Service (APNS) to implement push messaging. The application aims to provide a management tool that allows information to be retrieved about the web applications' traffic and then launch or stop cloud instances based on the current load. The use of push messaging was included to allow for the cloud-based services to send messages asynchronously back to the handset, thus eliminating the need for the client to wait for a response.

One of the topics investigated, as part of the benchmarking test we conducted, is energy consumption. Research in this area includes the work done by Flinn and Satyanarayanan [14]. They specifically look at energy-aware systems, where they show that a solution for dynamic balancing of energy consumption and application quality is an essential part of comprehensive energy management solutions. Similarly, Rivoire *et al*. [15] present a research effort in the area of energy efficiency. They propose *JouleSort*, an external sort benchmark for evaluating the energy efficiency of various devices, from laptops, desktops and servers.

The research described effort by Rivoire *et al*. [15] is quite different from ours. Firstly, we focus on one specific element, namely push messaging technologies running on an Android device. We chose to focus on the Android platform because it is well integrated with other Google technologies and, more importantly, it is an open platform making it ideal for experimentation. Secondly, we are not trying to identify the performance of disk I/O, CPU capacity and so on, but we compare the stability, response times and energy consumption of the important push messaging technologies. Generally, our test is much more specific, targeting specific mobile operating system and push-messaging technologies, in contrast to the general-purpose benchmark proposed by Rivoire *et al*. [15].

In the context of performance testing, Calheiros *et al*. [5] have examined the performance of cloud computing alternatives. They present a simulation toolkit, called *CloudSim*, making it possible to model and simulate

cloud computing systems and applications. They argue that it is impossible to perform benchmarking experiments in a repeatable, dependable and scalable environment using real-world cloud-based platforms. With CloudSim, they have created a tool making it fast and easy to configure and run these tests.

Nonetheless, getting highly reliable and repeatable results from a real-world environment is difficult [5]. This is especially so when using cloud-based resources, where there are many components, hardware and software, working together and it can be difficult to isolate the specific parts of the overall system one wants to test. However, we still believe there is value in doing a benchmarking test of different push messaging technologies integrated with cloud computing. There is value in providing performance results for the different technologies because it gives the developers information on the strengths and weaknesses of the popular push-messaging alternatives on the platform.

Also, it is important to note that we are integrating external components, the mobile devices, into the tests and not just using a cloud-based system, making it even more challenging to use a simulation tool that will provide realistic results.

We have focused our work on combining mobile devices with cloud computing. In this context, Binnig *et al*. [1] argue that a benchmarking test in a cloud computing environment should address the following issues:

1) *Adaptability of the system*, the ability to adapt to changing load in terms of scalability and cost;

2) *Conduct the benchmarking tests from different locations*;

3) *Access more dynamic "Web* 2.0"*-like applications including multimedia content.*

These pointers provide valuable insight into what a cloud computing benchmark test should include. We want to use some of the general ideas from this list, but we also want to stress the differences between what we are benchmarking, where a mobile client and cloud integration is involved, with the pure cloud-based benchmarking test that was described by Binnig *et al*. [1]. Particularly interesting for our test is to include the stability, response times and energy consumption of the system and also results from different networks.

The next section will introduce the different push messaging technologies.

3. Push Messaging on Android

The technologies used in our experiment all deal with the integration of cloud computing and mobile applications. Not all push messaging technologies investigated are directly related to cloud computing, such as XMPP (Extensible Messaging Presence Protocol). However, it is well integrated with the Google App Engine and is therefore included in our test.

In our benchmarking test we considered a total of six alternative technologies that offer push messaging for Android. These libraries are currently the main competitors in the market. SMS was not considered as part of the push messaging libraries, this was because of limitations such as availability and cost. Specifically, we conducted the test on a tablet device, Samsung Galaxy Tab 10.1, which does not support SMS.

The libraries we found particularly interesting are: *C2DM*, *Urban Airship*, *Xtify*, *XMPP*, *MQTT* (Message Queue Telemetry Transport) and *Deacon*. Although we believe these technologies present the most promising and useful push messaging libraries on Android, we cannot completely rule out the possibility of other interesting options we were not able to find.

Of these alternatives, we did not go into detail for two specific libraries, namely MQTT and Deacon. MQTT was not included because we wanted to investigate push-messaging technologies that can be easily integrated into the cloud, and specifically on the Google App Engine. MQTT is useful for connections that require a small code footprint and where network bandwidth is limited [16]. It does require a message broker hosted on a separate server. We did not find an easy way to integrate this service with the Google App Engine.

The second technology, The Deacon Project [17], is an open source project providing push notifications to Java and Android applications. We felt that this project was the least mature technology of the options we considered, as it is currently in beta release. The project also states that it is created for users wanting to run push notifications on their own server and support Android versions lower than 2.2, whereas C2DM requires at least Android 2.2. None of these requirements matched what we wanted to investigate, which included a close integration with a cloud-based server application and devices running on at least the 2.2 version of Android.

3.1. XMPP

The first push-messaging technology we wanted to include in our benchmarking test was the XMPP protocol. It is created for real-time communication [18] and for streaming XML [19]. The technology behind XMPP was created in 1998 and then refined in the Jabber open source community in 1999 and 2000, before it was formalised by IETF (The Internet Engineering Task Force) in 2002 and 2003 [20]. It is commonly used in Instant Messaging (IM) and has been used by Google Talk, Jabber and other IM networks.

XMPP is offered as a service on the Google App Engine, making it possible to write cloud-based applications

on the Google infrastructure that is able to communicate with users or applications. Accordingly, we integrated with XMPP through the Google App Engine infrastructure. Our Android client used an XMPP library called asmack [20], which is a patched version of smack created for Android. Smack offers an XMPP library and is a pure Java implementation [21].

XMPP on the Google App Engine has a daily limit of 1 GB data sent and 100,000 invitations with the free default limit [22]. More resources can be purchased, with paid applications incurring a minimum spend of $2.10 per week.

3.2. Cloud to Device Messaging

Cloud to Device Messaging (C2DM) was made available from Android 2.2, where the goal was to make it easier for mobile applications to sync data with servers [23]. The technology is used in several standard Google applications including Gmail, Contacts and Calendar. When messages are received on the Android client, the system will wake up the application via an Intent broadcast, and pass the message data [13]. The message limit is set to 1024 bytes and developers are encouraged to send short messages, essentially notifying the mobile application that updated information can be retrieved from the server. C2DM is a free service, and the maximum number of messages that can be sent is approximately 200,000 per day [24]; however this can be increased if there is a need for more resources.

Google offers standard libraries for Android that makes it possible to use C2DM directly. **Figure 1** shows a basic overview of C2DM. We believe the C2DM library provides a good basis for a standard push messaging technology. However, we identified certain limitations with C2DM that we wanted to simplify and provide more features than the standard solution. This will be presented in more detail in a later section.

There will also be a market for different solutions, offering more comprehensive services and novel features not covered by the C2DM technology, such as a webpage for administration and multi-platform support. These features are targeted by other commercial technologies, such as Urban Airship and Xtify.

3.3. Urban Airship

Urban Airship provides a commercial option for sending push notifications on Android, Blackberry and the iOS platform [25]. It makes it easier for developers to create applications for multiple device types since it provides a single API for all the supported platforms. It consists of a library that is added to the project to hide all the low level complexity related to push messaging. In addition to push notifications, Urban Airship provides features such as rich push, push composer, reports, in-app purchase and subscriptions.

Urban Airship also offers a proprietary push-messaging platform called *Helium* that supports Android 1.6 and newer. With newer phones (minimum Android 2.2) it supports the use of C2DM. The *pro priceplan* costs $199 per month, and includes support for up to 10,000 users and unlimited push messages, with an additional $0.01 per user over this limit [26].

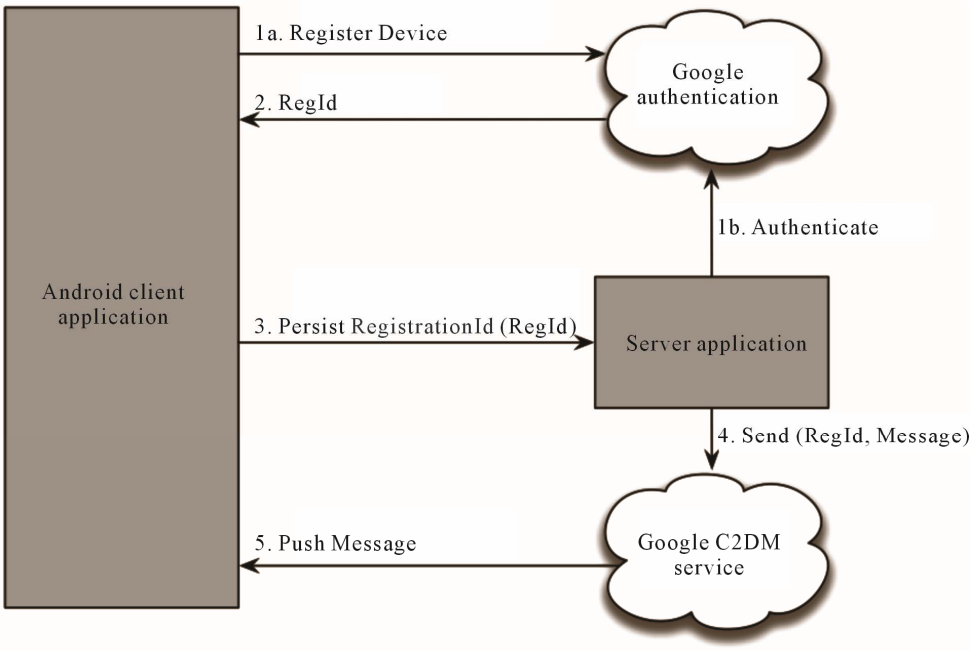

Figure 1. C2DM overview.

3.4. Xtify

Similar to Urban Airship, Xtify is a commercial option that provides push messaging for Android. Xitfy also supports the Android, Blackberry and iOS platforms. For Android it uses the C2DM technology offered by Google and adds features like registration management, notification handling, notification inbox, rich notification support and the ability to send messages based on user location.

The *Just Push*-package from Xtify costs $199 per month, and includes support for 30,000 devices and unlimited notifications. Each additional device over this limit costs $0.01 [27].

Both Urban Airship and Xtify support C2DM, and in addition they have their own proprietary Android push notification service [28]. These proprietary alternatives are used in the benchmarking test to provide a way of comparing C2DM-based applications with other push messaging technologies. For Xtify this is implemented on their infrastructure with the XMPP protocol [28]. It is important to note that even though two technologies in our benchmarking test are based on XMPP, we use XMPP directly when integrating with the Google App Engine. When testing Xtify we use their API and infrastructure. Xtify recommends using XMPP in cases where one for example needs to communicate frequently with the mobile devices over a short period of time. In other cases it recommends using the C2DM alternative it provides.

In our research reported here, we have included all of these technologies and performed a benchmarking test to see how they perform.

4. Benchmark Test

We created a benchmarking test to compare different push-messaging technologies on Android. The system consisted of a mobile client that in sequence invoked all the different push messaging technologies and record the time used. On the server side we have a Google App Engine server application that sends messages when requested to do so from the client. This application is also responsible for storing all the data received from the mobile application.

In our research we wanted to compare the performance of C2DM with other push-based technologies integrated in a cloud environment. We defined three main characteristics that are important for push-based technologies:

- **Response times**, what are the response times for the different push messaging technologies?
- **Stability**, are the response times providing stable results over the time we run the test?
- **Energy consumption**, how efficient, in regards to battery power, is the various push-messaging technologies?

The test was conducted in two main iterations. In iteration 1 we started by looking at the response times and stability for each push-based alternative. Moreover, the test was performed on two network types, namely WIFI and 3G. The message size sent was 450 bytes on all technologies and the tests were run on and off over several days with messages sent every 5 minutes.

For iteration 1 we included the following devices in the benchmarking test: *Samsung Galaxy Tab* 10.1 (SG), *HTC Evo* (HE), and *HTC Nexus One* (HN).

For iteration 2, we wanted to compare the energy consumption of the various push-messaging technologies. This test included the same message size (450 bytes), but we only used one device, which was SG, and we also increased the message frequency to 10 minutes. This device was selected to get a more comprehensive test because the SG device has a significantly larger battery (7000 mAh) when compared to for example the HE (1500 mAh).

This benchmark test lasted about 6 days for C2DM and Urban Airship, while the XMPP test only lasted about 2 days because of certain limitations in the platform, which we will describe in more detail later in this paper.

When doing our pilot-tests we noticed that the screen would consume a considerable amount of battery power and made it difficult to find any differences between the push-messaging technologies. For this reason we turned the screen on the device off when testing the energy consumption. Additionally, we also disabled the auto-sync feature to prevent applications using network communication resources on the device. These steps were taken to try to eliminate other factors that might impact the energy consumption on the device.

4.1. Test Procedure

Both iterations, as explained above, followed these main steps:

1) The Android client registers with the server. This is done differently for each technology, for example C2DM will send a registration id to the device;

2) A timer is started on the client, followed by a message being sent to the server requesting a new push message;

3) The server application receives the message and immediately sends out a message consisting of 450 bytes to the mobile device. This will happen for each technology type;

4) When the message is received, the Android client stops the timer and registers the result. This result is then sent back to a result-servlet that is part of the server application, which will permanently store the information;

5) Finally, the process waits 5/10 minutes before continuing with the sending the next message.

In our tests we compare the following technologies: 1) *XMPP*; 2) *Urban Airship Helium*; 3) *Xtify* (proprietary push messaging infrastructure) and 4) *C*2DM. All push messaging alternatives are integrated with a cloud-based Google App Engine server application. It is important to note that we did not include other technologies built on top of C2DM, like Urban Airship or Xtify with C2DM enabled, because they send messages in the same way as standard C2DM.

4.2. Results

Table 1 presents the overall results from the benchmarking test. We start by looking at the numbers for both the SG and HE. Both devices ran on the same WIFI network, and we were also able to provide a fairly equal number of messages for each technology providing a good basis for the comparison.

As can be seen in the table below, the results are relatively consistent, even though the differences between the technologies (see the standard deviation) were biggest on the SG. The average response time for XMPP was the shortest, with C2DM on second and finally Urban Airship. There is a difference of 276.12 ms (SG) and 156.98 (HE) between XMPP, which had the shortest response times, and Urban Airship with the longest response times. As seen in both **Figure 2** (results from SG) and 3 (results from HE), the difference is mostly due to spikes in the response from Urban Airship. The max time used for Urban Airship was 5337 ms (SG) and 3601 ms (HE), whereas both XMPP and C2DM provided considerably more stable results in the benchmarking test. These spikes were more frequent on the results gathered from SG than with HE, however, this trend was evident on all the devices included in the test, which is confirmed by looking at the standard deviation.

Overall the C2DM results were stable and the performance results recorded showed an average response time of 466.82 ms (SG) over 281 messages and 401.89 ms (HE) over 174 messages. Comparing the response time for C2DM on SG and HE, there is only a difference of 23.01 ms in the average response times. The HE had fewer messages received, with 174 compared to 281 for the tablet.

XMPP had the most stable results in our test, with a standard deviation of 172.91 (SG) and 67.84 (HE). Urban Airship did appear to have more stability issues than the rest and these issues surfaced several times during the test. When comparing the results from different WIFI networks and 3G, the same pattern emerges. The 3G response times are higher, but this is to be expected since they will have less bandwidth than the WIFI connection.

The final technology we tested was Xtify, and it comes very close to the overall performance of C2DM. It provides more stable results than Urban Airship and with slightly better average response time than C2DM. We were unable to conduct the benchmarking test with Xitfy on other devices than HE, because of limitations in the account we used. Additionally, since we were only able to send a limited number of messages this technology was not included in iteration 2.

The results for SG and HE are presented in **Figures 2** and **3** on the next page, and it is easy to the spikes in the response times for Urban Airship as previously mentioned.

For the final test, iteration 2, we wanted to investigate the energy consumption of the different push-messaging technologies. In this part of the benchmark test, we ran the same application as before, but we increased the time

Table 1. Test results.

Device	Tech	Number of messages	Average response time (ms)	Standard deviation
Samsung Galaxy Tab 10.1 —*Android* 3.1 —*WIFI*	C2DM	281	466.82	203.76
	Urban Airship	279	619.43	708.72
	XMPP	280	343.31	172.91
HTC Evo —*Android* 2.3 —*WIFI*	C2DM	174	401.89	95.40
	Urban Airship	172	473.88	321.97
	XMPP	168	316.90	67.84
HTC Nexus One —*Android* 2.3 —3*G*	C2DM	17	502.47	59.68
	Urban Airship	37	814.27	943.24
	XMPP	30	436.60	286.10
HTC Evo —*Android* 2.3 —*WIFI*	Xtify	213	432.92	250.09

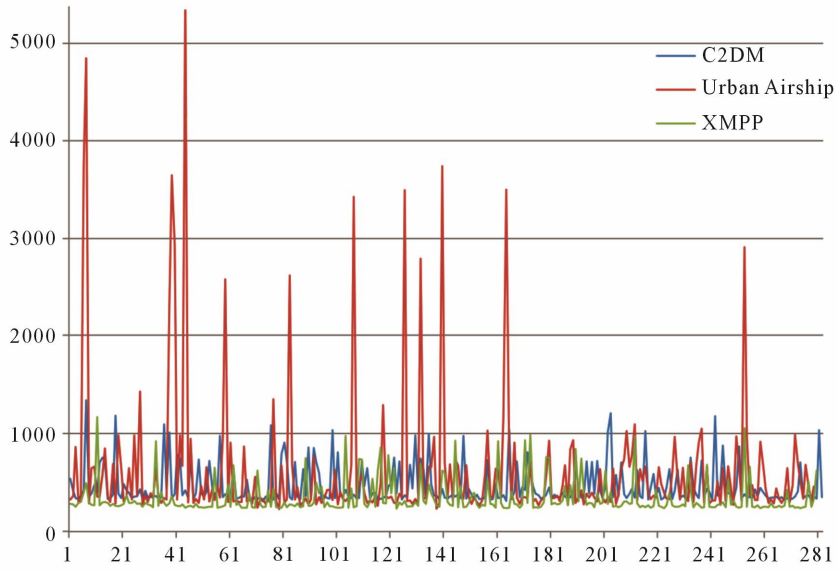

Figure 2. Results for C2DM, Urban Airship and XMPP on Samsung Galaxy Tab (WIFI).

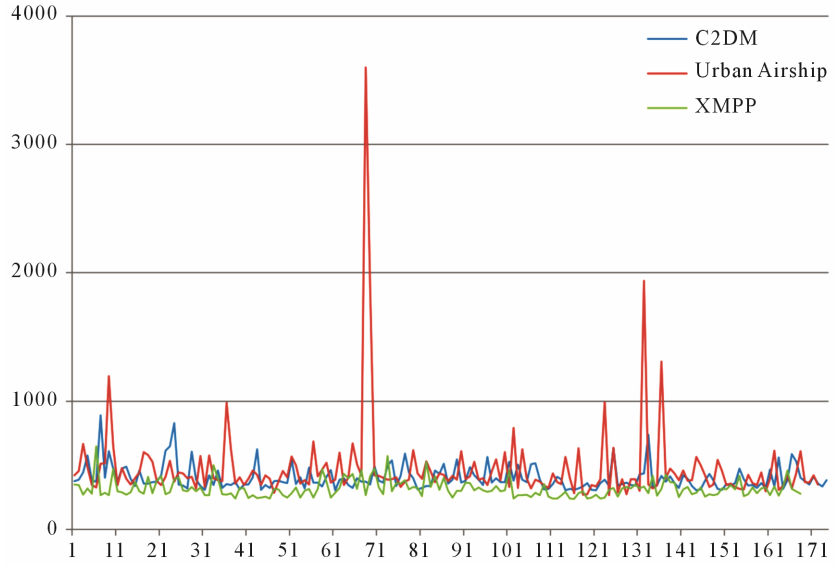

Figure 3. Results for C2DM, Urban Airship and XMPP on HTC Evo (WIFI).

between messages to 10 minutes. Another difference was that instead of running each technology in sequence, we only recorded one technology at a time. This was done because we wanted to provide results based on the battery level for each technology, and also to expand the test over a longer period of time. As previously described, the client ran the test by sending requests to the cloud-based server, but as part of this iteration we also added a feature that triggered a new request from the mobile device for each change in battery level. By doing this, we were able to record the messages and also the corresponding battery level.

In this test we included C2DM, Urban Airship, and XMPP. Xtify was not included due to limitations with the developer account we had created.

With both C2DM and Urban Airship we were able to provide fairly equal number of messages, with 862 and 858 messages sent respectively. However, with XMPP we were unable to send more than 295 messages because of quotas and limits in the Google App Engine [29]. The results are presented in **Figure 4**, where we have added trend lines for each result to make it easier to see the differences between the technologies.

As can be seen in **Figure 4**, both C2DM and Urban Airship provided the best results, using less energy than XMPP. We did expect this because it is recommended to use XMPP in scenarios where one needs to communicate frequently over a short period of time, as stated by Xtify

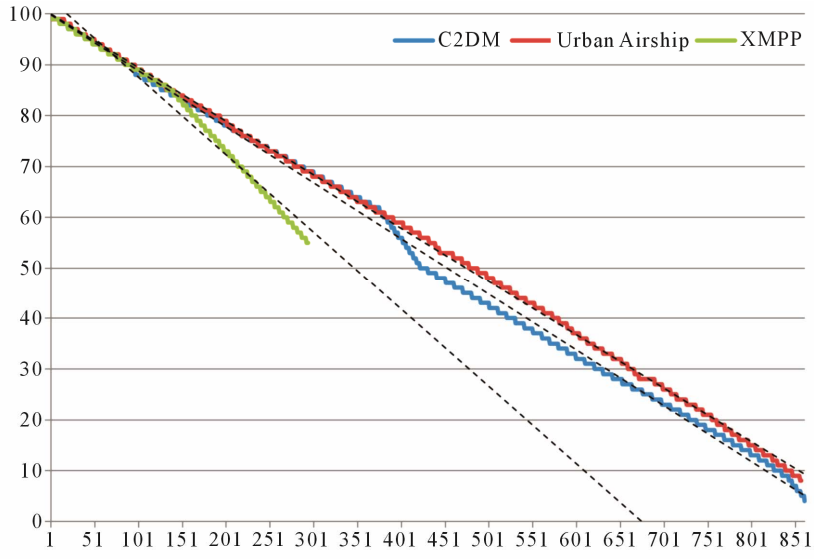

Figure 4. Energy consumption results.

[30]. This aspect is also verified by the normal use of XMPP, for instance Instant Messaging applications such as Google Talk.

The two other technologies, Urban Airship and C2DM, provided fairly similar results. However, from the last 50% and towards 0%, Urban Airship did provide slightly better results. It is difficult to draw any final conclusions of the difference between these technologies because the change is fairly small, and further testing is needed. However, we still find these results interesting, and especially the big difference between C2DM/Urban Airship and XMPP. We would certainly recommend using either C2DM or Urban Airship over XMPP for applications not dependent on frequent messages being pushed from the server to the devices.

Table 2 presents a short summary of the advantages and disadvantages of the libraries we tested. The overall results show that the C2DM, which is the standard Android library offered by Google, provided good performance compared to the alternatives. Moreover, it also performed well in the energy consumption test, especially compared to XMPP. We found that C2DM provided the best overall results in the three categories investigated, namely response time, stability and energy consumption.

4.3. Limitations

The benchmarking test, as described previously, has some limitations, and this was specifically apparent on some of the mobile devices. We had certain issues running all of the push messaging technologies over a period of time. This would result in certain messages not being received. The test would run reliably for period of time, before the messages were no longer registered on the mobile client. A restart of the Android client would solve the problem, but this scenario happened multiple times. This is why some of the devices have very few test results for certain technologies, for instance the HN had issues with C2DM. This only happened on the Android 2.3 mobile devices, whereas the Android 3.1 tablet was very stable over the entire test period. It would be interesting and useful for future work to focus on these stability issues, by including Android version 4 devices in the tests to see if the problems are fixed or at least improved in this newer version of the operating system. However, at the time of writing, an official version of Android 4 is not released for the devices used in the benchmark test.

Additionally, both Urban Airship and Xtify were tested on development servers. In the case of Xtify, we had to run this separately because there is a limit of 300 messages sent when using a basic account. The test consisted of a total of 213 messages, because we had to setup and test the system before running the actual experiment. For the energy consumption test we were unable to include Xtify because of the limitation in the number of messages.

Moreover, we used a fairly coarse grained scale, with battery percentage, when registering the energy consumption. There are other research efforts that have measured energy efficiency with a power meter, which is used in Rivoire *et al*. [15]. This will in most cases provide more reliable and accurate result. However, we believe there is still value in our general results and we were also able to push a fairly significant number of messages with: 862 (C2DM), 858 (Urban Airship) and 295 (XMPP).

We experienced a few issues with the C2DM API that we wanted to improve. These were specifically issues

Table 2. Result summary.

Technology	Advantages	Disadvantages
C2DM	The standard push-messaging technology offered by Google.	Certain features, such as sending messages to multiple clients, is not supported.
	Provided stable response times and low energy consumption compared to for example XMPP.	The development environment and API could be better (see Section 4.4 for more information on this issue).
Urban Airship	Provides several features not found in the free alternatives.	We recorded varying response times. In certain cases we saw quite a big increase in response times for some requests.
	Our results show that the energy consumption was considerably less than for XMPP.	Offers certain free features, but is mainly a commercial product.
XMPP	Works well in situations where one needs to communicate frequently over a short period of time.	Uses more battery power than the other technologies.
	Provided good and stable response times in our benchmark test.	
Xtify	For Xtify we were not able to collect the same amount of data, however, the results were very similar to C2DM in terms of response time.	Offers certain free features, but is mainly a commercial product.

related to the lack of flexibility and certain useful features. We created an open source library, called Simple-C2DM, to address these issues. We will give a short presentation of this library in the next section.

4.4. Simple-C2DM

We implemented a new open source library called Simple-C2DM. It was created specifically to simplify the development of applications using C2DM on the Android platform. Our library builds on top of standard C2DM and provides additional features.

An in-depth look at the Simple-C2DM features and functionality is outside the scope of this paper, however, a short introduction is given to provide useful information on how the development tools and API problems with C2DM can be improved. These features can also be useful for other push messaging technologies.

Figure 5 presents an overview of how Simple-C2DM is integrated in the client and server application.

There were two main reasons why we wanted to provide a new implementation:

1) To create a higher level of abstraction for certain key features;

2) To provide features not available in standard C2DM.

Starting with the higher level of abstraction, we solved this by introducing support for annotations. Annotations provide metadata that will not directly affect program semantics and are usually handled by tools and libraries. The main feature we found particularly useful and important by introducing annotation support was flexibility. Developers were no longer strictly forced to follow a specific pattern in the source code.

The second task we wanted to introduce in Simple-C2DM was new features not available in the standard implementation. One of the major issues with C2DM in our opinion is the *AndroidManifest.xml*-file, and especially the required configuration that needs to be provided. In Simple-C2DM we tried to solve this problem by offering two alternatives. If the developer does not want to, or cannot for some reason, use code generation, we created a manifest-generator hosted on the Google App Engine. This is a webpage that will take the package-name as input and generate the needed XML-tags.

The second option, which in our opinion is the best, is to generate the XML-tags automatically using an annotation processor. This part of the Simple-C2DM library was created as an experimental feature to figure out if we were able to automatically generate these XML-tags without the need to manually type in the package name. Because this uses an annotation processor, there is a configuration step involved when setting up the development environment. After this is completed, the annotation processor automatically runs when the project is compiled.

5. Conclusions

In this paper we investigated four cloud-integrated push-messaging technologies for the Android platform. We wanted to specifically target technologies that could be easily integrated with cloud computing, and in our test we used the Google App Engine as the cloud-based platform. We ran the tests on both WLAN and 3G, and provided results from several Android devices.

Our benchmarking test consisted of a client installed on the mobile devices and a server application running on the Google App Engine. The client had fixed time intervals where it would request a push message from the server. When this requested message was received, it recorded the time used. For the second part of our test, the energy consumption experiment, we used the same basic client but increased the time between each message

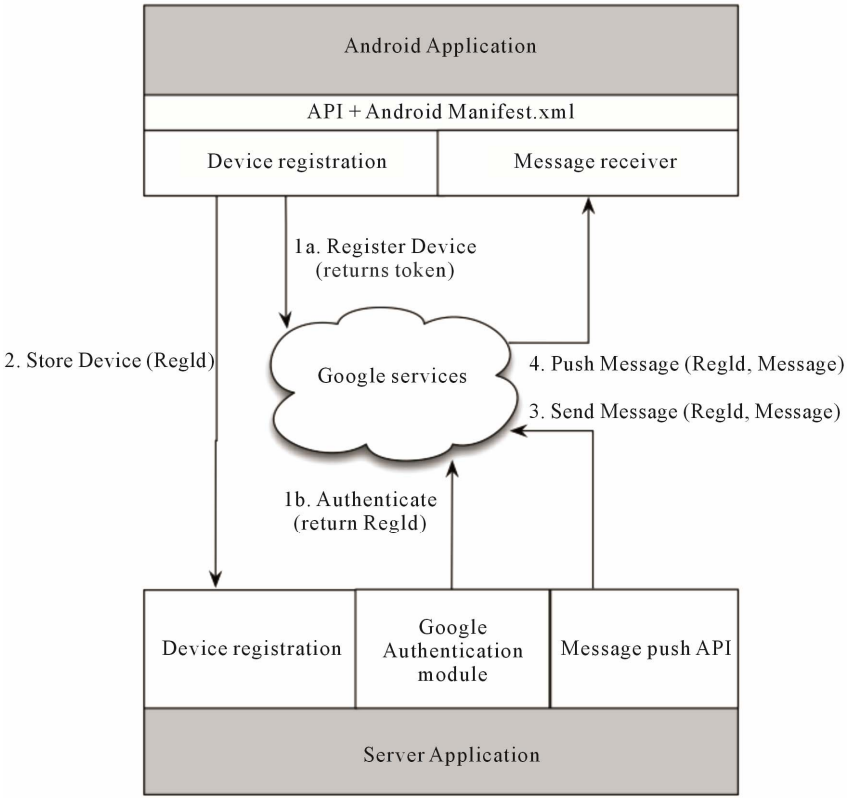

Figure 5. Simple-C2DM.

and included a feature that was able to record the battery level, in percentage, on the device.

The benchmarking test investigated each technology in regards to stability, response times and energy consumption. The results from the tests identified that XMPP provides the best result for response time and stability. However, it is also the library with the worst results for energy consumption. The next technology, Urban Airship, suffers from spikes in the response times, but does provide good results for the energy consumption test.

For Xtify we recorded very similar results to C2DM and Urban Airship with the stability and response times. However, we were not able to collect the same amount of data as with the other technologies, and were therefore unable to include the technology in the energy consumption test. Overall, when including all aspects of the test from the three technologies we were able to test thoroughly, we found that C2DM provides the best results. This is especially so if one does not need messages pushed frequently over a short period of time. In these cases XMPP also a good alternative, because it provided the best response times in our evaluation.

In future work we would like to investigate how the payload differs due to different message formats. Additionally, including devices with Android 4 in the benchmarking test can also be a useful extension to our work. Finally, conducting a more comprehensive energy consumption test, without the XMPP limitations and the inclusion of Xtify, would provide a useful future direction to our research.

REFERENCES

[1] C. Binnig, D. Kossmann, T. Kraska and S. Loesing, "How Is the Weather Tomorrow? Towards a Benchmark for the Cloud," *Proceedings of the 2nd International Workshop on Testing Database Systems*, ACM, New York, 2009, pp. 9:1-9:6.

[2] L. J. Mei, W. K. Chan and T. H. Tse, "A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues," *IEEE Asia-Pacific Services Computing Conference*, Yilan, 9-12 December 2008, pp. 464-469.

[3] Gartner, "Gartner Identifies the Top 10 Strategic Technologies for 2011," 2011. http://www.gartner.com/it/page.jsp?id=1454221

[4] Gartner, "Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 with 47 Percent Growth," 2012. http://www.gartner.com/it/page.jsp?id=1924314

[5] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software*: *Practice and Experience*, Vol. 41, No. 1, 2011, pp. 23-50. doi:10.1002/spe.995

[6] R. C. Elsenpeter, T. Velte and A. Velte, "Cloud Comput-

ing, A Practical Approach," McGraw-Hill Osborne Media, New York, 2009.

[7] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith and I. Sommerville, "The Cloud Adoption Toolkit: Supporting Cloud Adoption Decisions in the Enterprise," *Software*: *Practice and Experience*, Vol. 42, No. 4, 2012, pp. 447-465. doi:10.1002/spe.1072

[8] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[9] Google, "What Is Google App Engine?" 2011. http://code.google.com/appengine/docs/whatisgoogleappengine.html

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," 2009. http://www.eecs.berkeley.edu/Pubs/TechRpts/.../EECS-2009-28.pdf

[11] Google, "The Java Servlet Environment," 2012. https://developers.google.com/appengine/docs/java/runtime

[12] I. Podnar, M. Hauswirth and M. Jazayeri, "Mobile Push: Delivering Content to Mobile Users," *Proceedings of* 22*nd International Conference on Distributed Computing Systems Workshops*, Vienna, 2-5 July 2002, pp. 563-568.

[13] C. Paniagua, S. N. Srirama and H. Flores, "Bakabs: Managing Load of Cloud-Based Web Applications from Mobiles," *Proceedings of the* 13*th International Conference on Information Integration and Web-Based Applications and Services*, ACM, New York, 2011, pp. 485-490.

[14] J. Flinn and M. Satyanarayanan, "Managing Battery Lifetime with Energy-Aware Adaptation," *ACM Transactions on Computer Systems*, Vol. 22, No. 2, 2004, pp. 137-179. doi:10.1145/986533.986534

[15] S. Rivoire, M. A. Shah, P. Ranganathan and C. Kozyrakis, "JouleSort: A Balanced Energy-Efficiency Benchmark," *Proceedings of the* 2007 *ACM SIGMOD International Conference on Management of Data*, ACM, New York, 2007, pp. 365-376. doi:10.1145/1247480.1247522

[16] MQTT.org, "MQ Telemetry Transport," MQ Telemetry Transport, 2011. http://mqtt.org

[17] The Deacon Project, "The Deacon Project," 2012. http://deacon.daverea.com/about

[18] XMPP Standards Foundation, "About XMPP," 2011. http://xmpp.org/about-xmpp

[19] M. Ohja, "Server Push with Instant Messaging," *Proceedings of the* 2009 *ACM Symposium on Applied Computing*, ACM, New York, 2009, pp. 653-658.

[20] Asmack, "Asmack," 2012. http://code.google.com/p/asmack

[21] Smack, "Smack API," 2012. http://www.igniterealtime.org/projects/smack

[22] Google, "Quotas," 2012. http://code.google.com/appengine/docs/quotas.html

[23] Google, "Android Cloud to Device Messaging Framework," 2011. http://code.google.com/android/c2dm

[24] Google, "Android Cloud to Device Messaging: Quotas," 2012. http://code.google.com/android/c2dm/quotas.html

[25] Urban Airship, "Push Notifications," 2012. http://urbanairship.com/products/push-notifications

[26] Urban Airship, "Pricing," 2012. http://urbanairship.com/pricing

[27] Xtify, "Xtify Available Packages," 2012. http://www.xtify.com/pricing.html

[28] Xtify, "SDK Implementation Guides," 2012. http://developer.xtify.com/display/sdk/SDK+Implementation+Guides

[29] Google, "Quotas and Limits," 2012. https://developers.google.com/appengine/docs/java/xmpp/overview#Quotas_and_Limits

[30] Xtify, "Xtify Android XMPP Rich Notification Guide," 2012. http://developer.xtify.com/display/sdk/Xtify+Android+XMPP+Rich+Notification+Guide

